KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Module 1 Introduction to Software Engineering

PROFESSIONAL SOFTWARE ENGINEERING

Question	Answer
What is software?	Computer programs and associated documentation. Software products may be developed for a particular customer or may be developed for a general market.
What are the attributes of good software?	Good software should deliver the required functionality and performance to the user and should be maintainable, dependable and usable.
What is software engineering?	Software engineering is an engineering discipline that is concerned with all aspects of software production.
What are the fundamental software engineering activities?	Software specification, software development, software validation and software evolution.
What is the difference between software engineering and computer science?	Computer science focuses on theory and fundamentals; software engineering is concerned with the practicalities of developing and delivering useful software.
What is the difference between software engineering and system engineering?	System engineering is concerned with all aspects of computer-based systems development including hardware, software and process engineering. Software engineering is part of this more general process.
What are the key challenges facing software engineering?	Coping with increasing diversity, demands for reduced delivery times and developing trustworthy software.
What are the costs of software engineering?	Roughly 60% of software costs are development costs, 40% are testing costs. For custom software, evolution costs often exceed development costs.
What are the best software engineering techniques and methods?	While all software projects have to be professionally managed and developed, different techniques are appropriate for different types of system.
What differences has the web made to software engineering?	The web has led to the availability of software services and the possibility of developing highly distributed service-based systems. Web-based systems development has led to important advances in programming languages and software reuse.

Software Products

There are two kinds of software product:

1. *Generic products* These are stand-alone systems that are produced by a development organization and sold on the open market to any customer who is able to buy them. Examples of this type of product include apps for mobile devices, software for PCs such as databases, word processors, drawing packages, and project management tools. This kind of software also includes "vertical" applications designed for a specific market such as library information systems, accounting systems, or systems for maintaining dental records.

2. *Customized (or bespoke) software* These are systems that are commissioned by and developed for a particular customer. A software contractor designs and implements the software especially for that customer. Examples of this type of software include control systems for electronic devices, systems written to support a particular business process, and air traffic control systems.

Software Engineering

Software engineering is an engineering discipline that is concerned with all aspects of software production from the early stages of system specification through to maintaining the system after it has gone into use. In this definition, there are two key phrases:

1. *Engineering discipline* Engineers make things work. They apply theories, methods, and tools where these are appropriate. However, they use them selectively and always try to discover solutions to problems even when there are no applicable theories and methods. Engineers also recognize that they

must work within organizational and financial constraints, and they must look for solutions within these constraints.

2. *All aspects of software production* Software engineering is not just concerned with the technical processes of software development. It also includes activities such as software project management and the development of tools, methods, and theories to support software development.

Essential Attributes of Software Engineering

Product characteristic	Description
Maintainability	Software should be written in such a way so that it can evolve to meet the changing needs of customers. This is a critical attribute because software change is an inevitable requirement of a changing business environment.
Dependability and security	Software dependability includes a range of characteristics including reliability, security and safety. Dependable software should not cause physical or economic damage in the event of system failure. Malicious users should not be able to access or damage the system.
Efficiency	Software should not make wasteful use of system resources such as memory and processor cycles. Efficiency therefore includes responsiveness, processing time, memory utilisation, etc.
Acceptability	Software must be acceptable to the type of users for which it is designed. This means that it must be understandable, usable and compatible with other systems that they use.

Importance of Software Engineering

Software engineering is important for two reasons:

1. More and more, individuals and society rely on advanced software systems. We need to be able to produce reliable and trustworthy systems economically and quickly.

2. It is usually cheaper, in the long run, to use software engineering methods and techniques for professional software systems rather than just write programs as a personal programming project. Failure to use software engineering method leads to higher costs for testing, quality assurance, and long-term maintenance.

Software Process Activities

The systematic approach that is used in software engineering is sometimes called a software process. A software process is a sequence of activities that leads to the production of a software product. Four fundamental activities are common to all software processes.

1. Software specification, where customers and engineers define the software that is to be produced and the constraints on its operation.
2. Software development, where the software is designed and programmed.
3. Software validation, where the software is checked to ensure that it is what the customer requires.
4. Software evolution, where the software is modified to reflect changing customer and market requirements.

General Issues that affect Software

1. *Heterogeneity* Increasingly, systems are required to operate as distributed systems across networks that include different types of computer and mobile devices.

2. *Business and social change* Businesses and society are changing incredibly quickly as emerging economies develop and new technologies become available. They need to be able to change their existing software and to rapidly develop new software

3. *Security and trust* As software is intertwined with all aspects of our lives, it is essential that we can trust that software.

4. *Scale* Software has to be developed across a very wide range of scales, from very small embedded systems in portable or wearable devices through to Internet-scale, cloud-based systems that serve a global community.

Software Engineering ethics

It goes without saying that you should uphold normal standards of honesty and integrity. You should not use your skills and abilities to behave in a dishonest way or in a way that will bring disrepute to the software engineering profession. However, there are areas where standards of acceptable behavior are not bound by laws but by the more tenuous notion of professional responsibility. Some of these are:

1. *Confidentiality* You should normally respect the confidentiality of your employers or clients regardless of whether or not a formal confidentiality agreement has been signed.
2. *Competence* You should not misrepresent your level of competence. You should not knowingly accept work that is outside your competence.
3. *Intellectual property rights* You should be aware of local laws governing the use of intellectual property such as patents and copyright. You should be careful to ensure that the intellectual property of employers and clients is protected.
4. *Computer misuse* You should not use your technical skills to misuse other people's computers. Computer misuse ranges from relatively trivial (game playing on an employer's machine) to extremely serious (dissemination of viruses or other malware).

SOFTWARE PROCESS MODELS

1. *The waterfall model* This takes the fundamental process activities of specification, development, validation, and evolution and represents them as separate process phases such as requirements specification, software design, implementation, and testing.

2. *Incremental development* This approach interleaves the activities of specification, development, and validation. The system is developed as a series of versions (increments), with each version adding functionality to the previous version.

3. *Integration and configuration* This approach relies on the availability of reusable components or systems. The system development process focuses on configuring these components for use in a new setting and integrating them into a system.

Waterfall Model

The stages of the waterfall model directly reflect the fundamental software development activities:

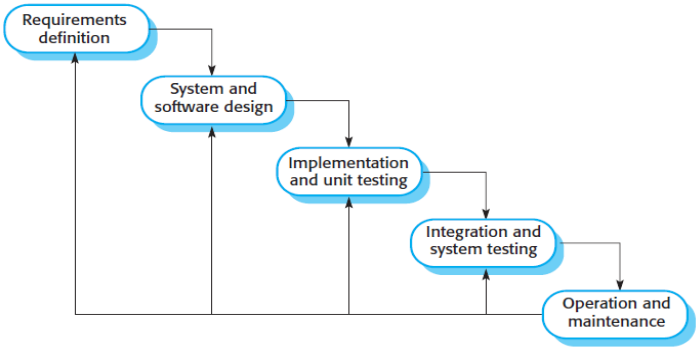
1. *Requirements analysis and definition* The system's services, constraints, and goals are established by consultation with system users. They are then defined in detail and serve as a system specification.

2. *System and software design* The systems design process allocates the requirements to either hardware or software systems. It establishes an overall system architecture. Software design involves identifying and describing the fundamental software system abstractions and their relationships.

3. *Implementation and unit testing* During this stage, the software design is realized as a set of programs or program units. Unit testing involves verifying that each unit meets its specification.

4. *Integration and system testing* The individual program units or programs are integrated and tested as a complete system to ensure that the software requirements have been met. After testing, the software system is delivered to the customer.

5. *Operation and maintenance* Normally, this is the longest life-cycle phase. The system is installed and put into practical use. Maintenance involves correcting errors that were not discovered in earlier stages of the life cycle, improving the implementation of system units, and enhancing the system's services as new requirements are discovered.



In principle, the result of each phase in the waterfall model is one or more documents that are approved ("signed off"). The following phase should not start until the previous phase has finished. The waterfall model is not the right process model in situations where informal team communication is possible and software requirements change quickly. Iterative development and agile methods are better for these systems.

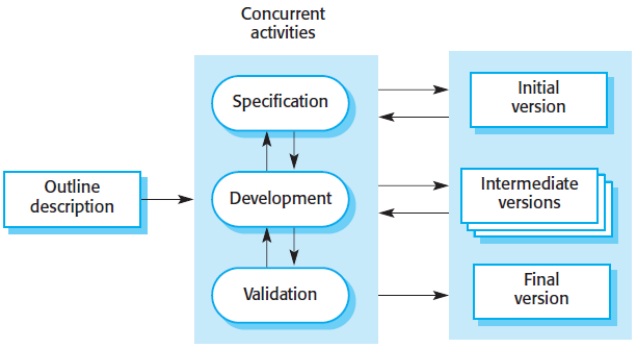
The waterfall model is only appropriate for some types of system:

1. Embedded systems where the software has to interface with hardware systems. Because of the inflexibility of hardware, it is not usually possible to delay decisions on the software's functionality until it is being implemented.

2. Critical systems where there is a need for extensive safety and security analysis of the software specification and design. In these systems, the specification and design documents must be complete so that this analysis is possible. Safety related problems in the specification and design are usually very expensive to correct at the implementation stage.

3. Large software systems that are part of broader engineering systems developed by several partner companies.

Incremental Development Model



- Incremental development is based on the idea of developing an initial implementation, getting feedback from users and others, and evolving the software through several versions until the required system has been developed. Specification, development, and validation activities are interleaved rather than separate, with rapid feedback across activities.

- Incremental development in some form is now the most common approach for the development of application systems and software products. This approach can be either plan-driven, agile or, more usually, a mixture of these approaches. In a plan-driven approach, the system increments are identified in advance; if an agile approach is adopted, the early increments are identified, but the development of later increments depends on progress and customer priorities.

- Each increment or version of the system incorporates some of the functionality that is needed by the customer. Generally, the early increments of the system include the most important or most urgently required functionality. This means that the customer or user can evaluate the system at a relatively early stage in the development to see if it delivers what is required. If not, then only the current increment has to be changed and, possibly, new functionality defined for later increments.

- Incremental development has three major advantages over the waterfall model:
  1. The cost of implementing requirements changes is reduced. The amount of analysis and documentation that has to be redone is significantly less than is required with the waterfall model.

  2. It is easier to get customer feedback on the development work that has been done. Customers can comment on demonstrations of the software and see how much has been implemented. Customers find it difficult to judge progress from software design documents.

  3. Early delivery and deployment of useful software to the customer is possible, even if all of the functionality has not been included. Customers are able to use and gain value from the software earlier than is possible with a waterfall process.

- From a management perspective, the incremental approach has two problems:
  1. The process is not visible. Managers need regular deliverables to measure progress. If systems are developed quickly, it is not cost effective to produce documents that reflect every version of the system.

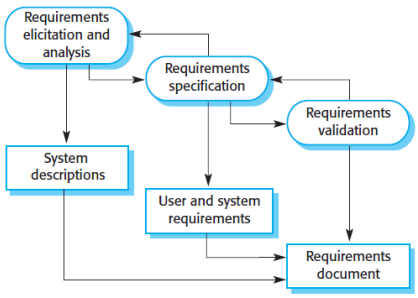
  2. System structure tends to degrade as new increments are added. Regular change leads to messy code as new functionality is added in whatever way is possible. It becomes increasingly difficult and costly to add new features to a system. To reduce structural degradation and general code messiness, agile methods suggest that you should regularly refactor (improve and restructure) the software.

PROCESS ACTIVITES

- The four basic process activities of specification, development, validation, and evolution are organized differently in different development processes. In the waterfall model, they are organized in sequence, whereas in incremental development they are interleaved.

Software Specifications

- Software specification or requirements engineering is the process of understanding and defining what services are required from the system and identifying the constraints on the system's operation and development.
- Requirements engineering is a particularly critical stage of the software process, as mistakes made at this stage inevitably lead to later problems in the system design and implementation



- There are three main activities in the requirements engineering process:

1. *Requirements elicitation and analysis* This is the process of deriving the system requirements through observation of existing systems, discussions with potential users and procurers, task analysis, and so on.

This may involve the development of one or more system models and prototypes. These help you understand the system to be specified.

2. *Requirements specification* Requirements specification is the activity of translating the information gathered during requirements analysis into a document that defines a set of requirements. Two types of requirements may be included in this document. User requirements are abstract statements of the system requirements for the customer and end-user of the system; system requirements are a more detailed description of the functionality to be provided.

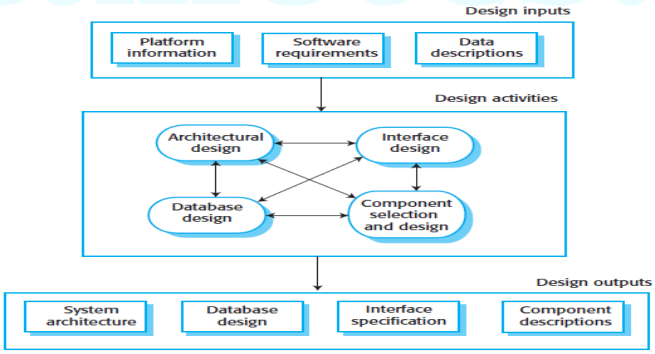
3. *Requirements validation* This activity checks the requirements for realism, consistency, and completeness. During this process, errors in the requirements document are inevitably discovered. It must then be modified to correct these problems.

- Requirements analysis continues during definition and specification, and new requirements come to light throughout the process. Therefore, the activities of analysis, definition, and specification are interleaved.

Software Design and Implementation

- The implementation stage of software development is the process of developing an executable system for delivery to the customer. Sometimes this involves separate activities of software design and programming. However, if an agile approach to development is used, design and implementation are interleaved, with no formal design documents produced during the process.

- A software design is a description of the structure of the software to be implemented, the data models and structures used by the system, the interfaces between system components and, sometimes, the algorithms used. Designers do not arrive at a finished design immediately but develop the design in stages. They add detail as they develop their design, with constant backtracking to modify earlier designs.



The activities in the design process vary, depending on the type of system being developed

1. *Architectural design*, where you identify the overall structure of the system, the principal components (sometimes called subsystems or modules), their relationships, and how they are distributed.
2. *Database design*, where you design the system data structures and how these are to be represented in a database. Again, the work here depends on whether an existing database is to be reused or a new database is to be created.
3. *Interface design*, where you define the interfaces between system components. This interface specification must be unambiguous
4. *Component selection and design*, where you search for reusable components and, if no suitable components are available, design new software components. The design at this stage may be a simple component description with the implementation details left to the programmer.

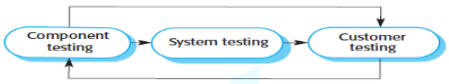
Software Validation

- Software validation or, more generally, verification and validation (V & V) is intended to show that a system both conforms to its specification and meets the expectations of the system customer.

- Program testing, where the system is executed using simulated test data, is the principal validation technique.
- Validation may also involve checking processes, such as inspections and reviews, at each stage of the software process from user requirements definition to program development. However, most V & V time and effort is spent on program testing.

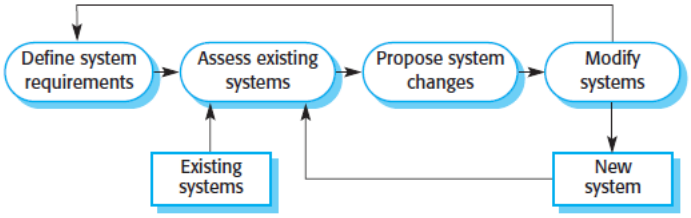
The stages in the testing process are:

1. *Component testing* The components making up the system are tested by the people developing the system. Each component is tested independently, without other system components. Components may be simple entities such as functions or object classes or may be coherent groupings of these entities.
2. *System testing* System components are integrated to create a complete system. This process is concerned with finding errors that result from unanticipated interactions between components and component interface problems. It is also concerned with showing that the system meets its functional and non-functional requirements, and testing the emergent system properties.
3. *Customer testing* This is the final stage in the testing process before the system is accepted for operational use. The system is tested by the system customer (or potential customer) rather than with simulated test data. For custom-built software, customer testing may reveal errors and omissions in the system requirements definition, because the real data exercise the system in different ways from the test data. Customer testing may also reveal requirements problems where the system's facilities do not really meet the users' needs or the system performance is unacceptable. For products, customer testing shows how well the software product meets the customer's needs.



Software Evolution

- The flexibility of software is one of the main reasons why more and more software is being incorporated into large, complex systems. Once a decision has been made to manufacture hardware, it is very expensive to make changes to the hardware design.
- However, changes can be made to software at any time during or after the system development. Even extensive changes are still much cheaper than corresponding changes to system hardware.



COPING WITH CHANGE

- Change is inevitable in all large software projects. The system requirements change as businesses respond to external pressures, competition, and changed management priorities. As new technologies become available, new approaches to design and implementation become possible. Therefore whatever software process model is used, it is essential that it can accommodate changes to the software being developed.

- Change adds to the costs of software development because it usually means that work that has been completed has to be redone. This is called rework. For example, if the relationships between the requirements in a system have been analyzed and new requirements are then identified, some or all of the requirements analysis has to be repeated. It may then be necessary to redesign the system to deliver the new requirements, change any programs that have been developed, and retest the system.

- Two related approaches may be used to reduce the costs of rework:

1. *Change anticipation*, where the software process includes activities that can anticipate or predict possible changes before significant rework is required.
2. *Change tolerance*, where the process and software are designed so that changes can be easily made to the system. This normally involves some form of incremental development.

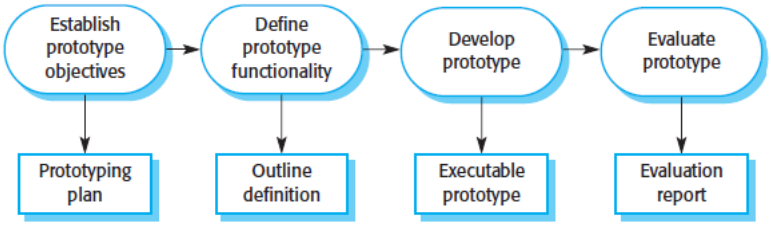
- two ways of coping with change and changing system requirements:

1. *System prototyping*, where a version of the system or part of the system is developed quickly to check the customer's requirements and the feasibility of design decisions. This is a method of change anticipation as it allows users to experiment with the system before delivery and so refine their requirements. The number of requirements change proposals made after delivery is therefore likely to be reduced.
2. *Incremental delivery*, where system increments are delivered to the customer for comment and experimentation. This supports both change avoidance and change tolerance. It avoids the premature commitment to requirements for the whole system and allows changes to be incorporated into later increments at relatively low cost.

System Prototyping

- A prototype is an early version of a software system that is used to demonstrate concepts, try out design options, and find out more about the problem and its possible solutions.
- Rapid, iterative development of the prototype is essential so that costs are controlled and system stakeholders can experiment with the prototype early in the software process.
- A software prototype can be used in a software development process to help anticipate changes that may be required:
  - In the requirements engineering process, a prototype can help with the elicitation and validation of system requirements.
  - In the system design process, a prototype can be used to explore software solutions and in the development of a user interface for the system.

- System prototypes allow potential users to see how well the system supports their work.
- They may get new ideas for requirements and find areas of strength and weakness in the software. They may then propose new system requirements.
- Furthermore, as the prototype is developed, it may reveal errors and omissions in the system requirements.



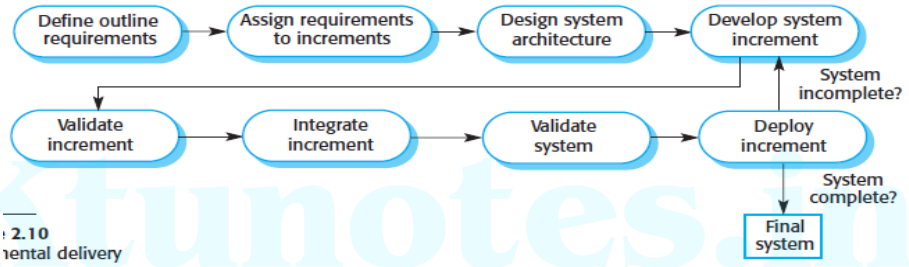
- The objectives of prototyping should be made explicit from the start of the process. These may be to develop the user interface, to develop a system to validate functional system requirements, or to develop a system to demonstrate the application to managers. The same prototype usually cannot meet all objectives

- The next stage in the process is to decide what to put into and, perhaps more importantly, what to leave out of the prototype system. To reduce prototyping costs and accelerate the delivery schedule, you may leave some functionality out of the prototype.

- The final stage of the process is prototype evaluation. Provision must be made during this stage for user training, and the prototype objectives should be used to derive a plan for evaluation. Potential users need time to become comfortable with a new system and to settle into a normal pattern of usage. Once they are using the system normally, they then discover requirements errors and omissions

Incremental Delivery

- Incremental delivery is an approach to software development where some of the developed increments are delivered to the customer and deployed for use in their working environment.
- In an incremental delivery process, customers define which of the services are most important and which are least important to them.
- A number of delivery increments are then defined, with each increment providing a subset of the system functionality.
- The allocation of services to increments depends on the service priority, with the highest priority services implemented and delivered first.
- Once the system increments have been identified, the requirements for the services to be delivered in the first increment are defined in detail and that increment is developed.
- During development, further requirements analysis for later increments can take place, but requirements changes for the current increment are not accepted.
- Once an increment is completed and delivered, it is installed in the customer's normal working environment.
- They can experiment with the system, and this helps them clarify their requirements for later system increments.
- As new increments are completed, they are integrated with existing increments so that system functionality improves with each delivered increment.



2.10
ıental delivery

Incremental delivery has a number of advantages:

1. Customers can use the early increments as prototypes and gain experience that informs their requirements for later system increments. Unlike prototypes, these are part of the real system, so there is no relearning when the complete system is available.
2. Customers do not have to wait until the entire system is delivered before they can gain value from it. The first increment satisfies their most critical requirements, so they can use the software immediately.
3. The process maintains the benefits of incremental development in that it should be relatively easy to incorporate changes into the system.
4. As the highest priority services are delivered first and later increments then integrated, the most important system services receive the most testing. This means that customers are less likely to encounter software failures in the most important parts of the system.

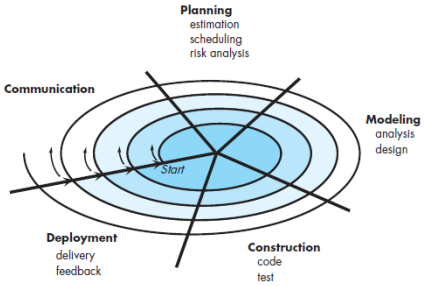
Key problems with this approach are:

1. Iterative delivery is problematic when the new system is intended to replace an existing system. Users need all of the functionality of the old system and are usually unwilling to experiment with an incomplete new system.
2. Most systems require a set of basic facilities that are used by different parts of the system. As requirements are not defined in detail until an increment is to be implemented, it can be hard to identify common facilities that are needed by all increments.
3. In the incremental approach, there is no complete system specification until the final increment is specified. This requires a new form of contract, which large customers such as government agencies may find difficult to accommodate.

BOHEM'S SPIRAL MODEL

- Originally proposed by Barry Boehm the *spiral model* is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model. It provides the potential for rapid development of increasingly more complete versions of the software

- A spiral model is divided into a set of framework activities defi ned by the software engineering team.



Each cycle in the spiral is divided into four parts:

Objective setting: Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.

Risk Assessment and reduction: The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.

Development and validation: The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.

Planning: Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

When to use Spiral Model?

- When deliverance is required to be frequent.
- When the project is large
- When requirements are unclear and complex
- When changes may require at any time
- Large and high budget projects

Advantages

- High amount of risk analysis
- Useful for large and mission-critical projects.

Disadvantages

- Can be a costly model to use.
- Risk analysis needed highly particular expertise
- Doesn't work well for smaller projects

AGILE SOFTWARE DEVELOPMENT

- Agile methods are incremental development methods in which the increments are small, and, typically, new releases of the system are created and made available to customers every two or three weeks.
- They involve customers in the development process to get rapid feedback on changing requirements. They minimize documentation by using informal communications rather than formal meetings with written documents.
- Agile approaches to software development consider design and implementation to be the central activities in the software process.
- They incorporate other activities, such as requirements elicitation and testing, into design and implementation.

Agile Methods

- Dissatisfaction with the overheads involved in software design methods of the 1980s and 1990s led to the creation of agile methods. These methods:
    - Focus on the code rather than the design
    - Are based on an iterative approach to software development
    - Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- The aim of agile methods is to reduce overheads in the software process (e.g. by limiting documentation) and to be able to respond quickly to changing requirements without excessive rework.

Agile Manifesto:

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more†.

Principles of Agile Software Development

Principle	Description
Customer involvement	Customers should be closely involved throughout the development process. Their role is provide and prioritize new system requirements and to evaluate the iterations of the system.
Incremental delivery	The software is developed in increments with the customer specifying the requirements to be included in each increment.
People not process	The skills of the development team should be recognized and exploited. Team members should be left to develop their own ways of working without prescriptive processes.
Embrace change	Expect the system requirements to change and so design the system to accommodate these changes.
Maintain simplicity	Focus on simplicity in both the software being developed and in the development process. Wherever possible, actively work to eliminate complexity from the system.

Agile Principles Applicability

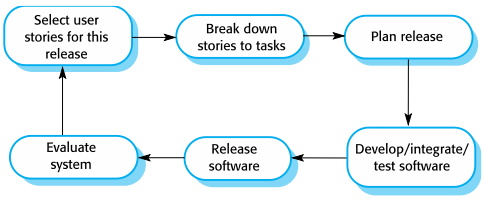
- Product development where a software company is developing a small or medium-sized product for sale. Virtually all software products and apps are now developed using an agile approach

- Custom system development within an organization, where there is a clear commitment from the customer to become involved in the development process and where there are few external rules and regulations that affect the software.

Agile Development Techniques

1. Extreme Programming:

- A very influential agile method, developed in the late 1990s, that introduced a range of agile development techniques.
- Extreme Programming (XP) takes an 'extreme' approach to iterative development.
  - New versions may be built several times per day;
  - Increments are delivered to customers every 2 weeks;
  - All tests must be run for every build and the build is only accepted if tests run successfully.



XP release cycle

Practices:

Principle or practice	**Description**
Incremental planning	Requirements are recorded on story cards and the stories to be included in a release are determined by the time available and their relative priority. The developers break these stories into development 'Tasks'. See Figures 3.5 and 3.6.
Small releases	The minimal useful set of functionality that provides business value is developed first. Releases of the system are frequent and incrementally add functionality to the first release.
Simple design	Enough design is carried out to meet the current requirements and no more.
Test-first development	An automated unit test framework is used to write tests for a new piece of functionality before that functionality itself is implemented.
Refactoring	All developers are expected to refactor the code continuously as soon as possible code improvements are found. This keeps the code simple and maintainable.
Pair programming	Developers work in pairs, checking each other's work and providing the support to always do a good job.
Collective ownership	The pairs of developers work on all areas of the system, so that no islands of expertise develop and all the developers take responsibility for all of the code. Anyone can change anything.
Continuous integration	As soon as the work on a task is complete, it is integrated into the whole system. After any such integration, all the unit tests in the system must pass.
Sustainable pace	Large amounts of overtime are not considered acceptable as the net effect is often to reduce code quality and medium term productivity
On-site customer	A representative of the end-user of the system (the customer) should be available full time for the use of the XP team. In an extreme programming process, the customer is a member of the development team and is responsible for bringing system requirements to the team for implementation.

XP and Agile Principles

- Incremental development is supported through small, frequent system releases.
- Customer involvement means full-time customer engagement with the team.

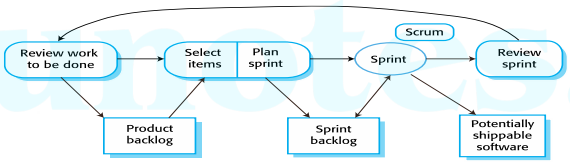
- People not process through pair programming, collective ownership and a process that avoids long working hours.
- Change supported through regular system releases.
- Maintaining simplicity through constant refactoring of code.

Agile Project Management

- The principal responsibility of software project managers is to manage the project so that the software is delivered on time and within the planned budget for the project.
- The standard approach to project management is plan-driven. Managers draw up a plan for the project showing what should be delivered, when it should be delivered and who will work on the development of the project deliverables.
- Agile project management requires a different approach, which is adapted to incremental development and the practices used in agile methods.

SCRUM

- Scrum is an agile method that focuses on managing iterative development rather than specific agile practices.
- There are three phases in Scrum.
  - The initial phase is an outline planning phase where you establish the general objectives for the project and design the software architecture.
  - This is followed by a series of sprint cycles, where each cycle develops an increment of the system.
  - The project closure phase wraps up the project, completes required documentation such as system help frames and user manuals and assesses the lessons learned from the project.



Scrum Sprint Cycle/ Scrum Process

- Sprints are fixed length, normally 2–4 weeks.
- The starting point for planning is the product backlog, which is the list of work to be done on the project.
- The selection phase involves all of the project team who work with the customer to select the features and functionality from the product backlog to be developed during the sprint.
- Once these are agreed, the team organize themselves to develop the software.
- During this stage the team is isolated from the customer and the organization, with all communications channelled through the so-called ‘Scrum master’.
- The role of the Scrum master is to protect the development team from external distractions.
- At the end of the sprint, the work done is reviewed and presented to stakeholders. The next sprint cycle then begins.

Team Work in Scrum

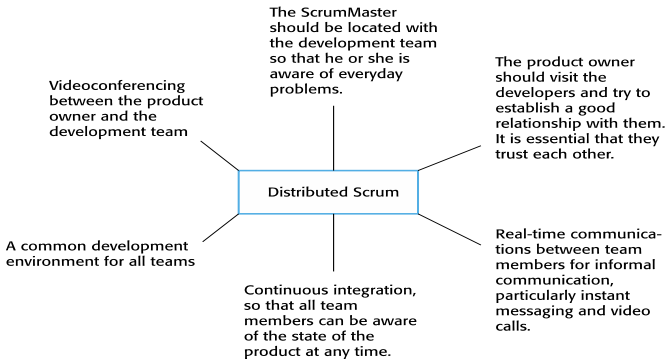
- The ‘Scrum master’ is a facilitator who arranges daily meetings, tracks the backlog of work to be done, records decisions, measures progress against the backlog and communicates with customers and management outside of the team.
- The whole team attends short daily meetings (Scrums) where all team members share information, describe their progress since the last meeting, problems that have arisen and what is planned for the following day.

- This means that everyone on the team knows what is going on and, if problems arise, can re-plan short-term work to cope with them.

Scrum Benefits

- The product is broken down into a set of manageable and understandable chunks.
- Unstable requirements do not hold up progress.
- The whole team have visibility of everything and consequently team communication is improved.
- Customers see on-time delivery of increments and gain feedback on how the product works.
- Trust between customers and developers is established and a positive culture is created in which everyone expects the project to succeed.

Distributed Scrum



Scrum term	**Definition**
Development team	A self-organizing group of software developers, which should be no more than 7 people. They are responsible for developing the software and other essential project documents.
Potentially shippable product increment	The software increment that is delivered from a sprint. The idea is that this should be 'potentially shippable' which means that it is in a finished state and no further work, such as testing, is needed to incorporate it into the final product.
Product backlog	This is a list of 'to do' items which the Scrum team must tackle.
Product owner	An individual (or possibly a small group) whose job is to identify product features or requirements, prioritize these for development and continuously review the product backlog to ensure that the project continues to meet critical business needs. The Product Owner can be a customer but might also be a product manager in a software company or other stakeholder representative.
Scrum	A daily meeting of the Scrum team that reviews progress and prioritizes work to be done that day. Ideally, this should be a short face-to-face meeting that includes the whole team.
ScrumMaster	The ScrumMaster is responsible for ensuring that the Scrum process is followed and guides the team in the effective use of Scrum. He or she is responsible for interfacing with the rest of the company and for ensuring that the Scrum team is not diverted by outside interference..
Sprint	A development iteration. Sprints are usually 2-4 weeks long.
Velocity	An estimate of how much product backlog effort that a team can cover in a single sprint. Understanding a team's velocity helps them estimate what can be covered in a sprint and provides a basis for measuring improving performance.